I1-D3

First-Version Rule Markup Languages

Project number:	IST2004506779
Project title:	Reasoning on the Web with Rules and Semantics
Project acronym:	REWERSE
Document type:	D (deliverable)
Nature of document:	R (report)
Dissemination level:	PU (public)
Document number:	IST506779/Eindhoven/I1-D3/D/PU/b1
Responsible editor(s):	Gerd Wagner
Reviewer(s):	José Júlio Alves Alferes
Contributing participants:	Eindhoven, Lisbon
Contributing workpackages:	I1
Contractual date of delivery:	28. February 2005
Actual date of delivery:	21. April 2005

Abstract

This report discusses the design of rule markup languages on the basis of RDF, OWL, and the Semantic Web Rule Language (SWRL), as well as on the basis of the RuleML proposal. Our aim is to develop a general Web rule language framework. Rule concepts are defined with the help of MOF/UML, a subset of the UML class modeling language proposed by the Object Management Group (OMG) for the purpose of 'meta-modeling', i.e. for defining languages conceptually on the level of an abstract (semi-visual) syntax. From these MOF/UML language models we can obtain a concrete markup syntax by applying a mapping procedure for generating corresponding XML schemas.

Keyword List

rule markup languages, rule meta-models

Project co-founded by the European Commission and the Swiss Federal Office for Education and Science within the Sixth Framework Programme

First-Version Rule Markup Languages♥♥

Gerd Wagner[1], Carlos Viegas Damásio[2], Sergey Lukichev[3]

[1] Institute of Informatics, Brandenburg University of Technology at Cottbus
Email: G.Wagner@tu-cottbus.de

[2] Departamento de Informática, Fac. Ciências e Tecnologia, Universidade Nova de Lisboa, Portugal
Email: cd@di.fct.unl.pt

[3] Institute of Informatics, Brandenburg University of Technology at Cottbus
Email: slukichev@.tu-cottbus.de

Abstract

This report discusses the design of rule markup languages on the basis of RDF, OWL, and the Semantic Web Rule Language (SWRL), as well as on the basis of the RuleML proposal. Our aim is to develop a general Web rule language framework. Rule concepts are defined with the help of MOF/UML, a subset of the UML class modeling language proposed by the Object Management Group (OMG) for the purpose of 'meta-modeling', i.e. for defining languages conceptually on the level of an abstract (semi-visual) syntax. From these MOF/UML language models we can obtain a concrete markup syntax by applying a mapping procedure for generating corresponding XML schemas.

Table of Contents

♥♥ This work is based on valuable discussions with, and input from, the co-chairs of the RuleML initiative, Harold Boley and Said Tabet, who pioneered the area of "rules on the Web".

1. Introduction

Rule markup languages will be the vehicle for using rules on the Web and in distributed systems. They allow deploying, executing, publishing and communicating rules on the Web. They may also play the role of a lingua franca for exchanging rules between different systems and tools.

In a narrow sense, a rule markup language is a concrete (XML-based) rule syntax for the Web. In a broader sense, it should have an abstract syntax as a common basis for defining various concrete languages serving different purposes. The main purposes of a rule markup language is to permit reuse, interchange and publication of rules.

Our MOF/UML-based approach to the design of rule markup languages is concept-driven instead of syntax-driven. Consequently, it supports maintenance, modularity and reuse in the definition of syntaxes for specific user communities and for special purposes.

We define metamodels for vocabulary elements, derivation rules, integrity rules, reaction rules and production rules, which are compatible with all important concepts of OWL, SWRL and RuleML. These metamodels define our language REWERSE Rule Markup Language (R2ML). We can obtain concrete markup languages from these metamodels by mapping them to corresponding XML schemas with help of a general mapping procedure to be developed.

We do not follow the mathematical logic approach and the "axiom terminology" of OWL. Rather, we adopt a computational logic approach in order to be able to make the computational distinction between a definition and a constraint.

In this report we only develop the abstract syntax of our rule languages. In a companion report, I1-D3-2, we will define a model-theoretic semantics based on partial logic.

1.1. Rule Modeling and Markup – an Example

An example, where a derived attribute in a UML class model is defined by a derivation rule, is the following:

> *A car is available for rental if it is not assigned to any rental contract and does not require service.*

This rule defines the derived Boolean-valued attribute `isAvailable` of the class `RentalCar` by means of an association `isAssignedTo` between cars and rental contracts and the stored Boolean-valued attribute `requiresService`, as shown in the UML class diagram in Figure 1.

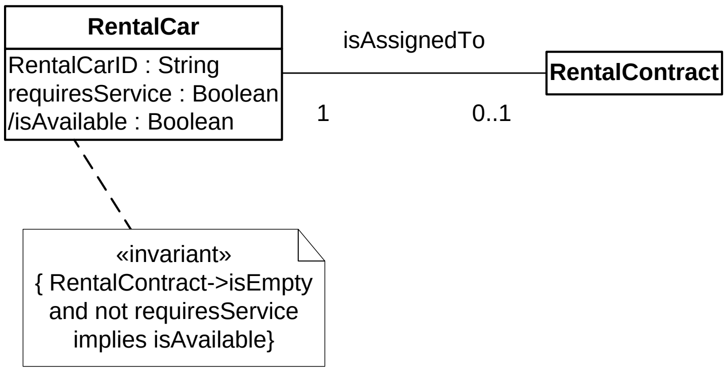

Figure 1: An OCL invariant that constrains the derived attribute *isAvailable*.

This class diagram specifies a vocabulary fragment consisting of

- two *basic entity types* (classes), `RentalCar` and `RentalContract`
- one *attribution fact type* that can be verbalized as: `RentalCar` has `String` as `RentalCarID`;
- two *subtypes* of `RentalCar`, `AvailableRentalCar` and the derived subtype `RentalCarRequiringService`, both being represented by Boolean-valued attributes
- an *association fact type*: `RentalCar isAssignedTo RentalContract`, which comes with three integrity rules and a clarification:
  1. *Functional*: It is necessary that each RentalCar isAssignedTo at most one RentalContract.
  2. *Inverse Total*: It is necessary that each RentalContract is assigned at least one RentalCar.
  3. *Inverse functional*: It is necessary that each RentalContract is assigned at least one RentalCar
  4. *Not total*: It is possible that a RentalCar isAssignedTo no RentalContract

An implicational OCL invariant, attached to the `RentalCar` class rectangle, is used to state that for a specific rental car whenever there is no rental contract associated with it, and it does not require service, then it must be available (for a new rental). In this OCL invariant expression, the condition `RentalContract->isEmpty()` means that the set of associated rental contracts must be empty.

However, such an OCL invariant does not really define anything but rather puts a constraint on the model elements it refers to. OCL 2.0, in addition to expressing integrity rules ('invariants'), also allows to express derivation rules for defining derived elements of a class model. Using this possibility, we get the following OCL expression:

```
  context RentalCar::isAvailable : Boolean derive:
  RentalContract->isEmpty() and not requiresService
```

This OCL derivation rule assigns the truth value of the conjunction

```
  RentalContract->isEmpty() and not requiresService
```

to the Boolean attribute `isAvailable` of the class `RentalCar`, and in this way it is a definition and not just a constraint.

We now present the concrete XML syntax of this rule according to the RuleML 0.88 syntax. Notice that the *head* element corresponds to the *conclusion*, and the *body* element corresponds to the *condition* of the derivation rule format defined in Figure 15. It is assumed that the attribute `requiresService` is optional, that is it does not need to have a value (in case it is unknown whether a particular car requires service or not). By contrast, the attribute `isAvailable` is assumed to be mandatory.

The first condition of this rule, `RentalContract->isEmpty()`, since it requires that there is no rental contract associated to the rental car, corresponds to a negation-as-failure, which is expressed by the tag `<naf>` in RuleML The second condition, `not requiresService`, corresponds to a strong negation since it requires that the value of this partial Boolean attribute is explicitly `false`. If it would be unknown, its negation with `not` would result in unknown and not in true.

So, this rule involves two kinds of negation, marked up with `<Naf>` and `<Neg>` in RuleML:

```
<Implies>
  <head>
    <Atom>
```

```
      <Rel>isAvailable</Rel>
      <Var>Car</Var>
    </Atom>
  </head>
  <body>
    <Atom>
      <Rel>RentalCar</Rel>
      <Var>Car</Var>
    </Atom>
    <Neg>
      <Atom>
        <Rel>requiresService</Rel>
        <Var>Car</Var>
      </Atom>
    </Neg>
    <Naf>
      <Atom>
      <Rel>isAssignedToRentalContract</Rel>
      <Var>Car</Var>
      </Atom>
    </Naf>
  </body>
</Implies>
```

1.2. Rule Concepts at Different Abstraction Levels

The Model Driven Architecture (MDA)[1] is a framework for model-driven software development defined by the OMG. MDA can be regarded as an evolutionary approach to bringing models as first-class citizens into the software engineering process. It is based on a fundamental distinction between three different modeling levels: the level of semi-formal ('computation-independent') business domain modeling, the level of platform-independent logical design modeling, and the level of platform-specific implementation modeling.

As illustrated in Figure 2, we consider rules at three different abstraction levels:

1. At the ('computation-independent') ***business domain level*** (called ***CIM*** in OMG's MDA), rules are statements that express (certain parts of) a business/domain policy (e.g., defining terms of the domain language or defining/constraining domain operations) in a declarative manner, typically using a natural language or a visual language. Examples are:

   (R1) “The driver of a rental car must be at least 25 years old”

   (R2) “A gold customer is a customer with more than $1Million on deposit”

   (R3) “An investment is exempt from tax on profit if the stocks have been bought more than a year ago”

   (R4) “When a share price drops by more than 5% and the investment is exempt from tax on profit, then sell it”
   R1 is an *integrity rule,* R2 and R3 are *derivation rules,* and R4 is a *reaction rule* (see below for explanations of these rule categories). These appear to be the major semantic categories of business rules. Actually, many business rules appear to be reaction rules, which specify policies for real-world business behavior.
2. At the platform-independent ***operational design level*** (called ***PIM*** in OMG's MDA), rules are formal statements, expressed in some formalism or computational paradigm, which can be

[1]See http://www.omg.org/cgi-bin/doc?mda-guide.

directly mapped to executable statements of a software system. Examples of rule languages at this level are SQL:1999, OCL 2.0, and DOM Level 3 Event Listeners. Remarkably, SQL provides operational constructs for all three business rule categories mentioned above: *checks/assertions* operationalize a notion of integrity rules, *views* operationalize a notion of derivation rules, and *triggers* operationalize a notion of reaction rules.

3. At the platform-specific ***implementation level*** (called ***PSM*** in OMG's MDA), rules are statements in a language of a specific execution environment, such as Oracle 10g views, Jess 3.4, XSB 2.6 Prolog, or the Microsoft Outlook 6 Rule Wizard.

Generally, rules are self-contained knowledge units that typically involve some form of reasoning. They may, for instance, specify

– static or dynamic integrity constraints (e.g. for constraining the state space or the execution histories of a system),
– derivations (e.g. for defining derived concepts),
– reactions (for specifying the reactive behavior of a system in response to events)

Given the linguistic richness and the complex dynamics of business domains, it should be clear that any specific mathematical account of rules, such as classical logic Horn clauses, must be viewed as a limited descriptive theory that captures just a certain fragment of the entire conceptual space of rules, and not as the only definitive, normative account. Rather, we need a pluralistic approach to the heterogeneous conceptual space of rules. Therefore, the goal of the REWERSE working group I1 is to define a family of rule languages capturing the most important types of rules. While these languages should come with a recommended standard semantics, their rule expressions may, in addition, allow alternate semantics, which are also considered acceptable. This will accommodate various formalisms based on non-standard logics, supporting temporal, fuzzy, defeasible, and other forms of reasoning.

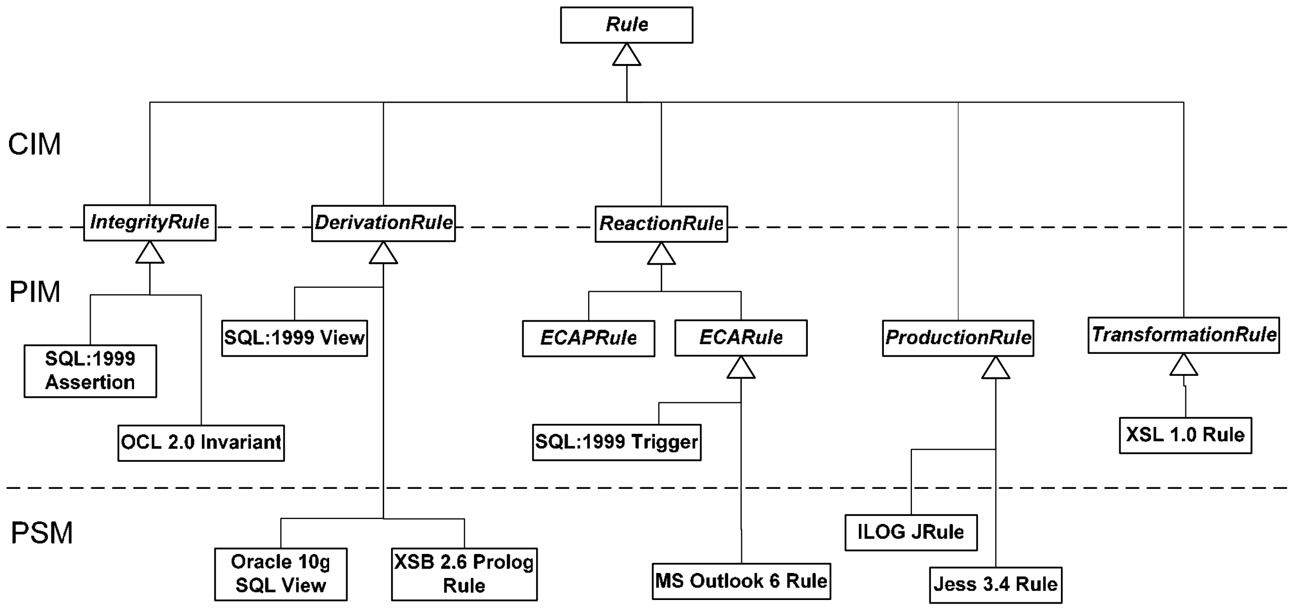

Figure 2: Rule concepts and rule expressions at different levels of abstraction.

The abstract syntax of a language can be defined with the help of a MOF/UML model, as recommended by the OMG, or it can be defined with the help of a suitably general grammar definition language such as the EBNF formalism used in the definition of the abstract syntax of

OWL and SWRL[2]. In our approach we prefer MOF/UML language definitions since they are more concise.

1.3. Categorizing Rules

The main categories of rules considered in this report are derivation rules, integrity rules (constraints), reaction rules, production rules and transformation rules, as depicted in Figure 2. We consider the concepts of derivation rules, integrity constraints, and reaction rules to be meaningful both as (computation-independent) business rule categories and as (platform-independent) computational rule categories, whereas the concepts of production rules and transformation rules appear to be only meaningful as computational rule categories.

Notice that those categories whose name is in italics, such as DerivationRule, refer to an abstract concept of rule, while the others (with non-italicized names), such as "SQL:1999 View", refer to rule concepts of concrete languages such as SQL:1999.

Each part of a rule is an expression of some type, playing a specific role within the rule. The different parts of rules named by the roles they play, and their types, are listed in Table 1. Table 2 shows the semantic categories denoted by the expression types mentioned in Table 1.

The main link between the different types of rules is the notion of a LogicalFormula or of a LogicalSentence, one of which being used in all of them. Traditionally, logical formulas are expressed in a language based on a predicate logic signature. However, OCL is the language of choice for expressing logical formulas referring to the state of a system whose structure is defined by a UML class model.

Table 1: Rule expression components and their corresponding types.

Rule Expression Component	*Type*	*Occurs in*
Condition	Logical Formula	DR, PR, RR
Conclusion	Logical Formula	DR
Post-Condition	Logical Formula	PR, RR
Produced Action	Action Expression	PR
Triggering Event	Event Expression	RR
Triggered Action	Action Expression	RR
Transformation Source	Term	TR
Transformation Target	Term	TR

Table 2: Semantic categories of the different expression types of rule parts

Type	*Semantic Category*
Logical Sentence	Truth value
Logical Formula	Function from variable bindings to truth values
Term	Can denote anything from some algebra
Event Expression	Event

[2] SWRL is the Semantic Web Rule Language -- see http://www.w3.org/Submission/SWRL/

Action Expression	Action

1.4. The REWERSE I1 Approach to Rule Markup Language Design

We assume that Web rule languages do not directly follow the tradition of predicate-logic-style rule languages such as Prolog, but rather follow the recent developments of Web knowledge representation languages such as RDF and OWL. This requires that they accommodate

1. Web naming concepts, such as URIs and XML namespaces,
2. the ontological distinction between objects (or individuals) and data values, and
3. the datatype concepts of RDF.

For simplicity, we follow OWL and SWRL, and the RuleML dialect NafNegDatalog, in not considering functions in this report. As a working name for the markup languages to be developed, we use the acronym R2ML standing for REWERSE Rule Markup Language.

Figure 3 is a simple example of how the abstract syntax of a formal expression is defined with MOF/UML. In this case, a traditional predicate logic atom is defined to consist of a predicate symbol and a sequence of arguments being (possibly complex) predicate logic terms.

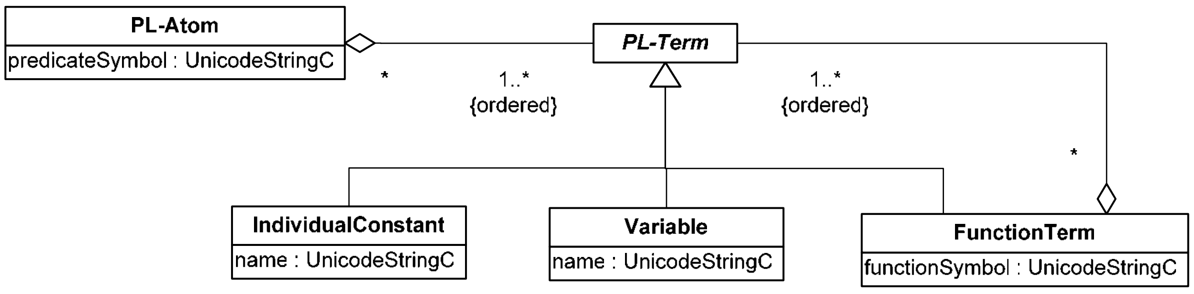

Figure 3: The abstract syntax of predicate logic atoms.

Such a metamodel can be transformed into a DTD or XML Schema definition by following a certain mapping procedure, like the one recommended by the XMI specification of the OMG.[3] In the case of the PL-Atom expression defined in Figure 3, a possible DTD-style language definition would be described by

```
ENTITY %PL-Term "( IndividualConstant | Variable | FunctionTerm )"

PL-Atom
    attributes: predicateSymbol
    content model: ( %PL-Term + )

FunctionTerm
    attributes: functionSymbol
    content model: ( %PL-Term + )

IndividualConstant
    attributes: name

Variable
    attributes: name
```

This example illustrates our approach. The DDT was obtained by

[3] See http://www.omg.org/technology/documents/formal/xmi.htm

1. mapping the segmentation defined by the abstract class *PL-Term* to a corresponding DDT choice assigned to a parameter entity (for making it available as a content model shared by PL-Atom and FunctionTerm)
2. mapping each non-abstract class to a corresponding XML element by
    a. mapping the attributes of the class to corresponding attributes of the element, taking into consideration if they are mandatory or optional
    b. mapping the parts of the class to corresponding subelements, taking the multiplicity constraints of the aggregation into consideration (e.g. 1..* maps to +)

In this report, we will define our rule markup languages primarily by means of MOF/UML models and then apply some mapping rules to obtain the corresponding DTD or XML schema of the markup language. This approach requires

1. to develop a style of MOF/UML modeling that is suitable for the purpose of defining formal languages for expressing vocabularies and rules; and
2. to develop suitable mapping rules for generating the target markup language definitions in the form of DTDs and XML schemas

In this report we will focus on the first issue, developing suitable MOF/UML models, while we will investigate the second issue as one of the next steps in our research.

We do not follow the mathematical logic approach and the "axiom terminology" of OWL. Rather, we adopt the computational logic approach in order to be able to make the computational distinction between a definition and a constraint.

We attempt to accommodate all important language constructs from OWL, SWRL and RuleML. Since these languages are not based on a common foundational ontology and are using some incoherent terminology we first have to harmonize the terminological differences. The results of this harmonization attempt are presented in Table 3, Table 4, Table 5 and Table 6.

Table 3: Basic language constructs for vocabularies

REWERSE-I1	*OWL / SWRL*	*RuleML*
object (individual)	individual	individual
data value (data literal)	data literal	
entity type (entity classification predicate)	class	relation
datatype (data classification predicate)	data range	
attribute (attribution predicate)	datatype property	
reference property (binary association predicate)	object property	
association predicate	n.a.	

Table 4: Names for different kinds of facts.

REWERSE-I1	*OWL / SWRL*	*RuleML*
entity classification fact	class description atom	positional fact

data classification fact	datarange atom	
attribution fact	datatype property atom	
reference property fact	object property atom	
association fact	n.a.	
object description	individual description	slot fact

Table 5: Language constructs needed for rules

REWERSE-I1	*RuleML*	*SWRL*
data variable	variable	data variable
object variable		individual variable
data term	term	data term
object term		individual term

Table 6: Accommodating different rule concepts. SWRL does not distinguish between an integrity rule and a derivation rule. All SWRL rules are "axioms".

REWERSE-I1	*RuleML*	*SWRL*
integrity rule `<IntegrityRule>`	integrity rule	rule axiom `<Implies>`
derivation rule `<DerivationRule>`	derivation rule `<Implies>`	
reaction rule `<ReactionRule>`	reaction rule	n.a.
production rule `<ProductionRule>`	production rule	n.a.

2. Basic Constructs for Vocabularies

Logical formulas, logical statements and rules are expressed in various ways on the basis of a vocabulary, which includes

– definitions of proper names or globally unique object identifiers (such as URI references) standing for ***objects*** (or ***individuals***);
– definitions of terms standing for ***general concepts***, among which we distinguish *entity types* (classes) and *datatypes*;
– definitions of *fact type expressions* (or *predicate symbols*) standing for **fact types** (or *predicates* and *properties*); and
– *fact statements* (expressing ***facts***) instantiating fact type expressions, within the definition of individuals

In Web languages such as RDF and OWL, all these language elements do not have ordinary names. Instead, they have globally unique standard identifiers in the form of URI references.

2.1. Basic Constructs of R2ML for Vocabularies

In this subsection, we discuss the basic constructs of R2ML for defining vocabularies including various kinds of predicates and atoms for expressing fact types and facts.

2.1.1. Data Values

Figure 4 models the construct of an *RDF literal*, which defines what is a data value, as being either a *plain literal* (possibly associated with some natural language) or a *typed literal*, which is a string associated with a datatype.

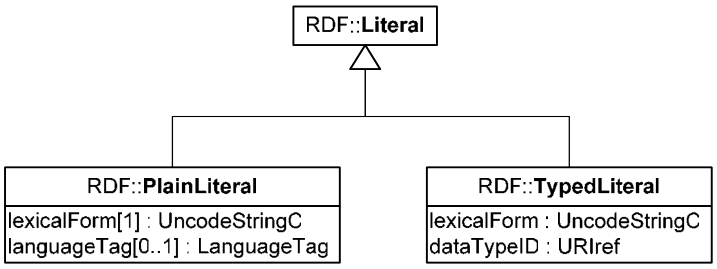
RDF::Literal
RDF::PlainLiteral
lexicalForm[1] : UncodeStringC
languageTag[0..1] : LanguageTag
RDF::TypedLiteral
lexicalForm : UncodeStringC
dataTypeID : URIref

Figure 4: An RDF data literal is either a plain literal, possibly associated with a language, or a typed literal.

2.1.2. Datatypes

The set rdfs:Literal is a datatype, and also each RDF datatype and each *DataLIteralEnumeration* is a datatype, as shown in Figure 5.

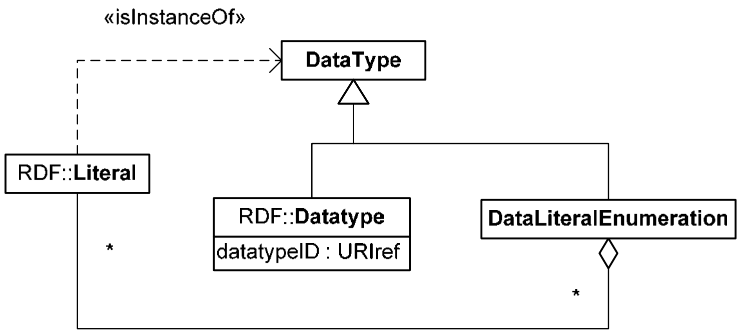

Figure 5: A datatype is an RDF datatype or a data literal enumeration, or it is equal to the set of RDF literals.

2.1.3. Attributes, Reference Properties and Object Descriptions

We also say *attribution predicate* instead of *attribute*, and *binary association predicate* instead of *reference property*. In OWL, an attribute is called 'datatype property' and a reference property is called 'object property'. Both are defined by means of 'property axioms'.

The concepts of attribute and reference property are defined in Figure 6 and Figure 7. In an OWL property axiom the definition of a property is mixed up with constraint assertions about it (such as requiring it to be functional). In the REWERSE-I1 approach, we keep definitions separate from constraints.

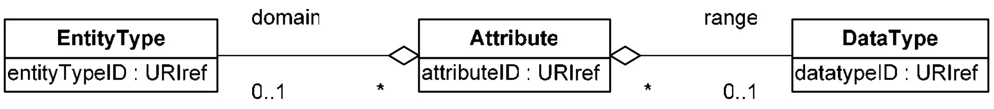

Figure 6: An attribute is defined with an entity type as its domain and a dtatype as its range.

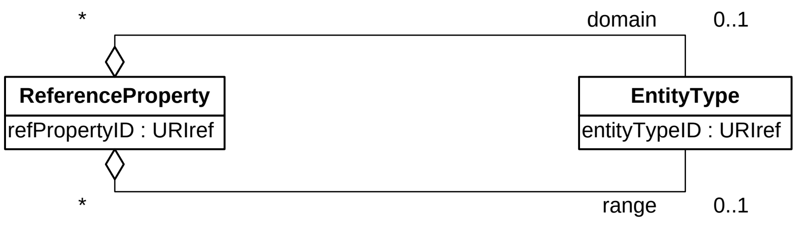

Figure 7: A reference property is defined with an entity type as its domain and an entity type as its range

Following RDF and OWL, we adopt the concept of an *object description fact*, as a collection of classification facts, attribution facts and reference property facts, all concerning one particular object that is referenced by an objectID, if it is not anonymous. This fact concept is depicted in Figure 8. In RDF and OWL, the much overloaded designation 'class' is used for entity type.

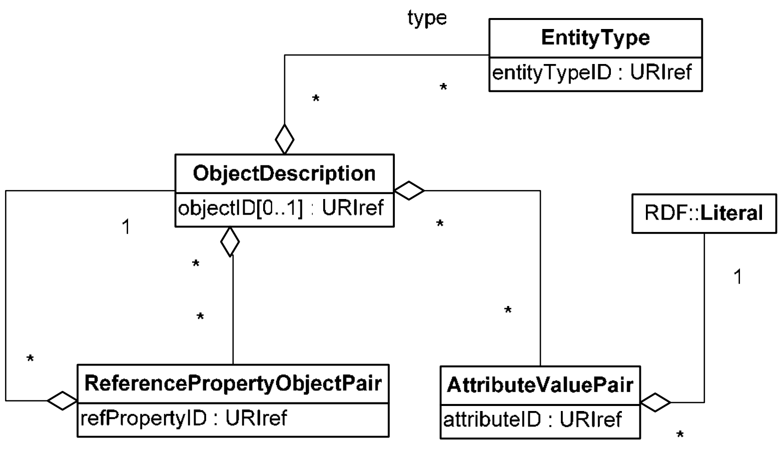

Figure 8: An object description fact aggregates object classification facts (assigning entity types to the 'subject'), attribution facts (assigning attribute-value pairs to the 'subject') and reference property facts (assigning reference-property-object pairs to the 'subject').

Notice that object classification facts, attribution facts and reference property facts are special cases of object description facts. So, we don't need to define them separately.

For defining the concept of an *object description fact type* (or *object description atom*), we first need to introduce the concept of a *logical variable*, as shown in Figure 9, where, as usual, the variable concept is subsumed under the concept of a term.

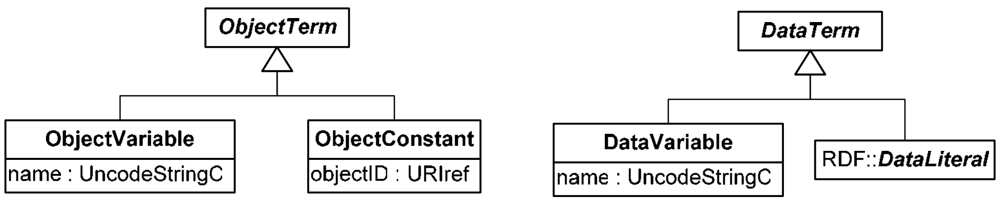

Figure 9: There are two kinds of logical terms: object terms and data terms.

Object description atoms are defined in Figure 10. Notice that we identify an entity type with the corresponding *object classification predicate*.

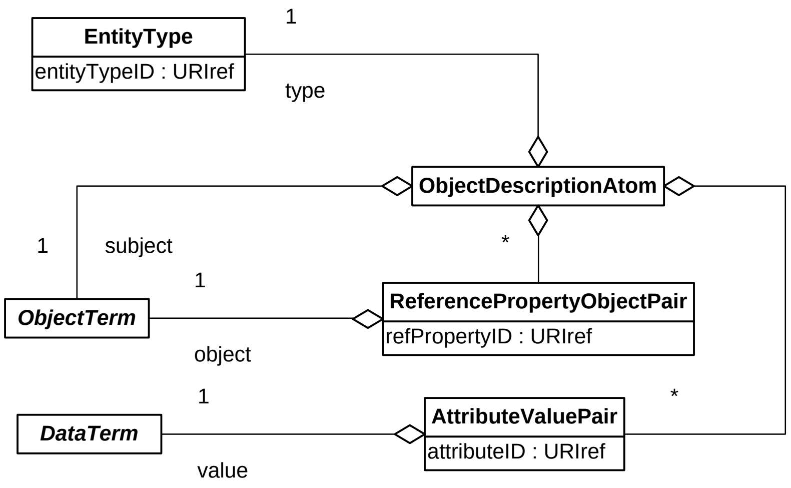

Figure 10: An object description atom is a shorthand for providing a set of descriptions for a particular object. It corresponds to a conjunction of a classification fact, a set of attribution facts and a set of reference property facts.

2.1.4. Association Predicates and Association Atoms

In order to directly support common fact types of natural language, it is important to have n-ary predicates for n>2 in the language. We therefore define the concept of an association predicate in Figure 11.

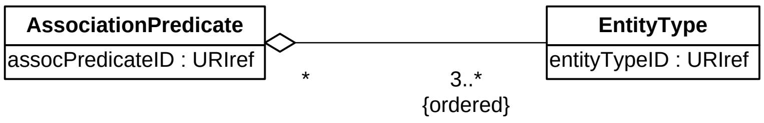

Figure 11: An association predicate associates three or more entity types.

We can form association atoms with the help of association predicates, as depicted in Figure 12.

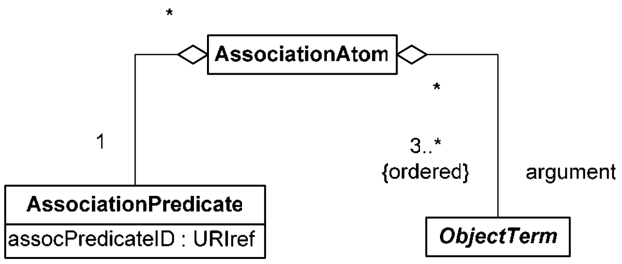

Figure 12: An association atom can express an n-ary association between classes.

2.1.5. Equality and Inequality Facts

As in OWL, we need to be able to express equality and inequality facts. For this purpose, we define the concept of an *equality atom* in Figure 13.

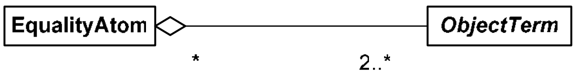

Figure 13: Equality atoms allow to express that two object identifiers designate the same individual.

The concept of an *inequality atom* is defined in the same way.

2.2. Taking a Quick Look at the Vocabulary Language OWL

In OWL, all vocabulary definitions, such as class definitions, datatype definitions and property definitions are 'axioms'. OWL has three other kinds of axioms which allow to express constraints: disjointness, equivalence and subsumption axioms.

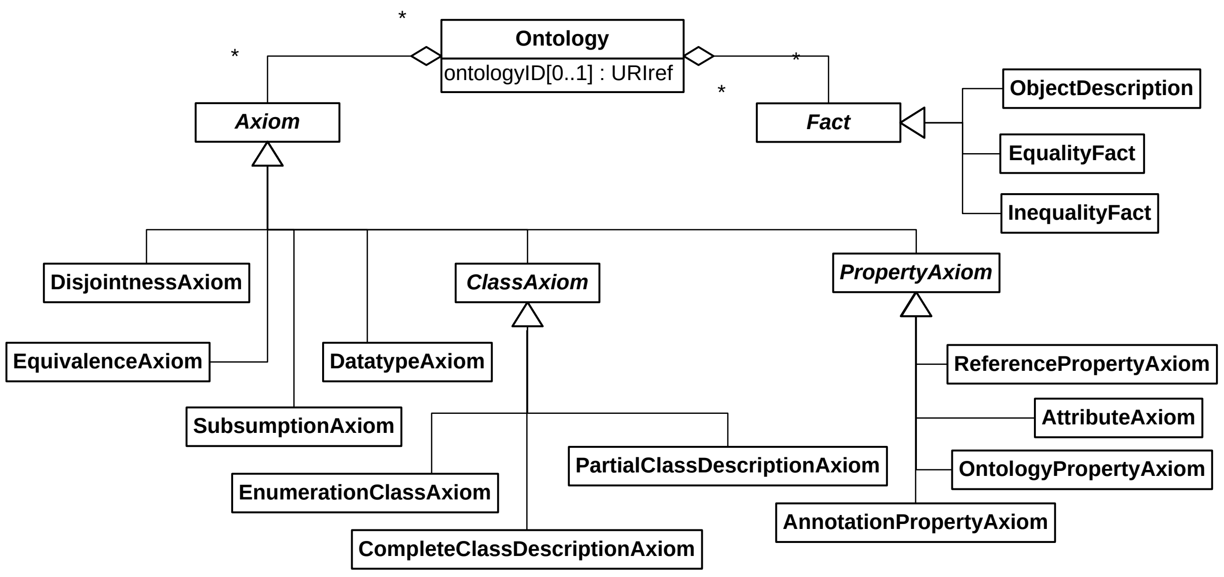

Figure 14: An OWL ontology consists of 'axioms' and facts.

3. Derivation Rules

Derivation rules, in general, consist of one or more *conditions* and one or more *conclusions*[4], which are both roles played by expressions of the type LogicalFormula.

For specific types of derivation rules, such as definite Horn clauses or normal logic programs, the types of condition and conclusion are specifically restricted.

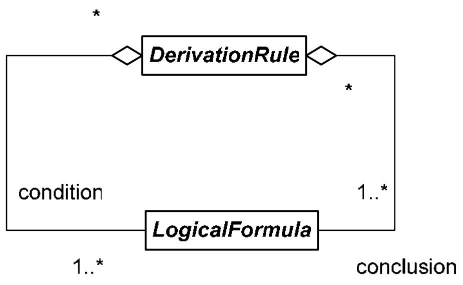

Figure 15: The abstract syntax of derivation rules.

3.1. The Semantic Web Rule Language (SWRL)

Figure 16 gives an overview of the abstract sytax of SWRL, which extends OWL by adding the concepts of variable and atom. In SWRL, variable names are URI references. Since variables are scoped to rule expressions, they do not need to be globally unique. Therefore, in R2ML variable names are strings, as shown in Figure 9.

[4] Notice that we don't consider rules with no condition or no conclusion. These expressions are better not called "rules", but "facts" and "denial constraints".

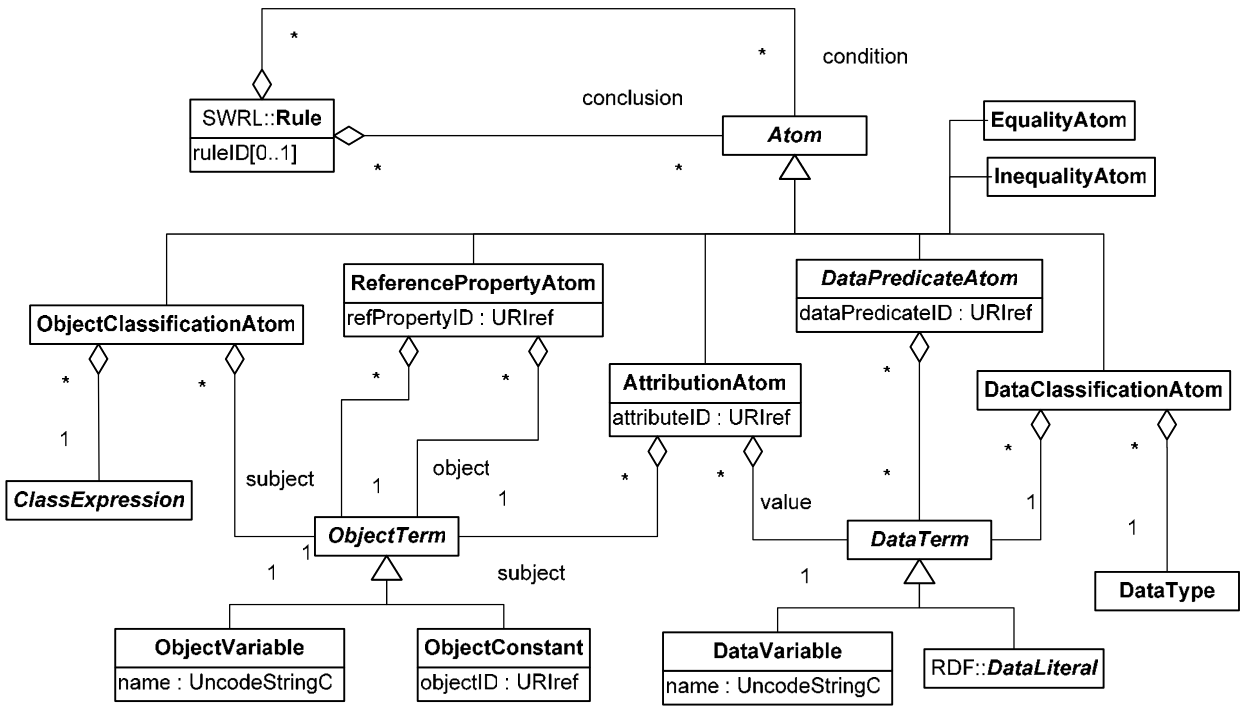

Figure 16: The abstract syntax of SWRL rules.

SWRL does not include the convenience construct of an object description atom. Such atoms have to be expressed in SWRL by a set of object classification atoms, reference property atoms and attribution atoms, all referring to the same object.

SWRL allows data predicate atoms and data classification atoms not only in the condition, but also in the conclusion of a rule. It is not clear what such 'rules' would be good for, except for defining derived datatype predicates and datatypes. Formally, they may also act as constraints, but that doesn't seem to make any sense.

3.2. The Rule Markup Language (RuleML)

RuleML defines a family of rule markup languages in a modular way, using the modularization techniques of DTDs and XML Schema. This approach allows to provide

In RuleML 0.88, a rulebase or knowledge base consists of universally quantified atoms (as facts), derivation rules and atom equivalences, as depicted in Figure 17.

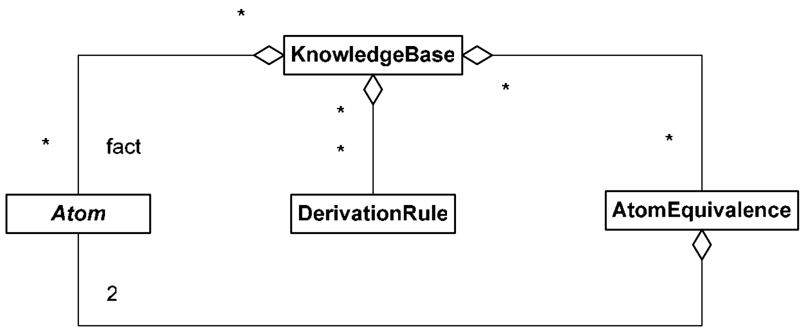

Figure 17: A RuleML 0.88 knowledge base consists of (universally quantified) atoms as facts, derivation rules and atom equivalence assertions.

The conclusion of a RuleML derivation rule, as depicted in Figure 18, is a neg-formula, that is, it is either an atom or a strongly negated atom, as in Figure 20.

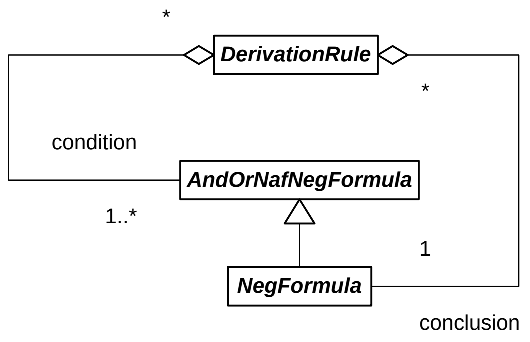

Figure 18: Derivation rules in RuleML 0.88

The conditions of a dervation rule are and-or-naf-neg-formulas as defined in Figure 19.

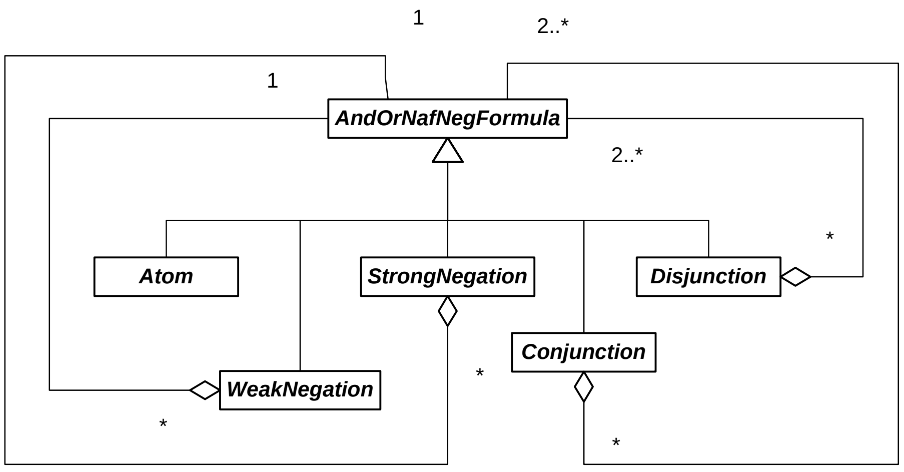

Figure 19: Quantifier-free formulas with weak and strong negation.

The distinction between weak and strong negation (marked up as <naf> and <neg> in RuleML) is present in several computational languages: in extended *logic programs* it is present in explicit form, while it is only implicitly present in SQL and OCL. Intuitively speaking, weak negation captures the absence of positive information, while strong negation captures the presence of explicit negative information (in the sense of Kleene's 3-valued logic). Under the preferential model semantics of minimal/stable models, weak negation captures the computational concept of negation-as-failure (or closed-world negation).

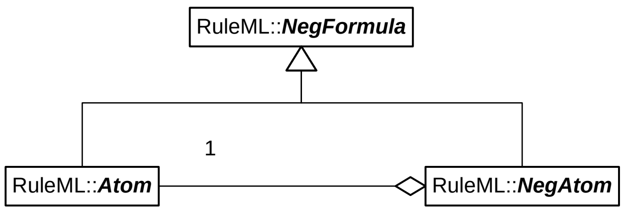

Figure 20: A neg-formula is either an atom or a strong negation of an atom.

In RuleML 0.88 there are two kinds of atoms (as depicted in Figure 21):

1. Prolog-style (predicate logic) atoms, called 'positional atoms',

2. object description atoms, called 'slot atoms'

Unlike in our object description atoms, the slot names of RuleML 'slot atoms' are 'individual constants'. This may be a problem for their semantics, since they correspond to attributes and reference properties (i.e. to predicate symbols) and not to individuals.

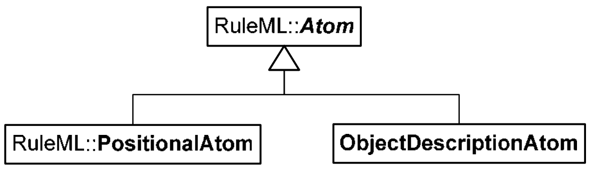

Figure 21: RuleML 0.88 admits of two kinds of atoms.

3.3. R2ML Derivation Rules

For derivation rules, we adopt the RuleML definition shown in Figure 18. However, we use a modified and extended definition of atom, where we replace the RuleML 'positional atom' with the concept of an association atom defined in Figure 12 and add the concept of data classification atom from SWRL. For conclusions we have to restrict the admissible atoms to *object atoms*. We thus obtain Figure 22 as a definition of R2ML atoms and object atoms.

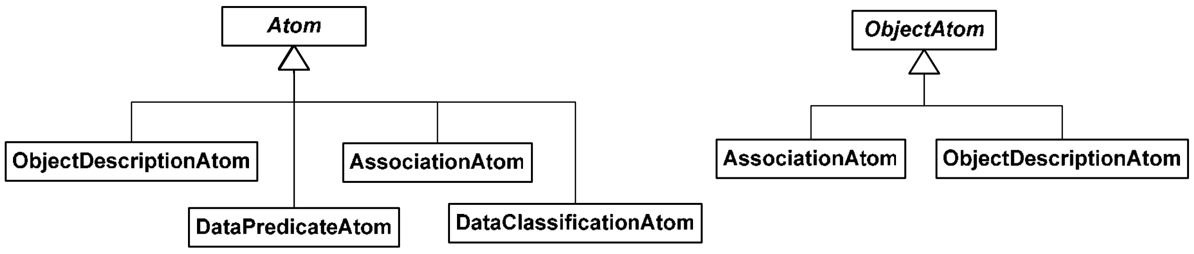

Figure 22: Object atoms and atoms in R2ML

The object atom concept is used in the definition of NegObjectFormula in Figure 23..

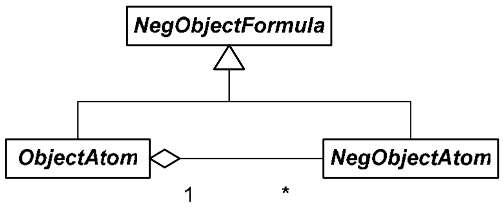

Figure 23: Neg-formula conclusions in R2ML

Finally, for R2ML we obtain the form of derivation rule depicted in Figure 24.

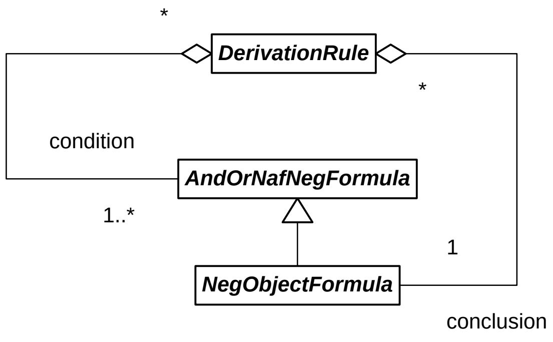

Figure 24: Derivation rules in R2ML

4. Integrity Rules (Constraints)

Integrity rules, also known as (integrity) constraints, consist of a constraint modality and a constraint assertion, which is a sentence in some logical language such as first-order predicate logic or OCL. This is depicted in Figure 25. We consider two constraint modalities: the ***alethic*** and the ***deontic*** one. The alethic constraint modality can be expressed by a phrase such as "it is necessarily the case that". The deontic constraint modality can be expressed by phrases such as "it is obligatory that" or "it should be the case that".

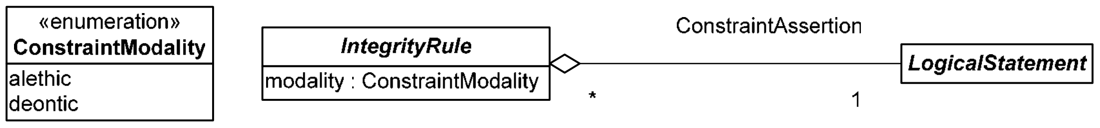

Figure 25: Abstract integrity rules

The constraint assertion is a logical sentence that *must necessarily*, or that *should*, hold in all evolving states and state transition histories of the discrete dynamic system to which it applies. Notice that not only software systems, but also physical, biological and social systems, such as organizations, can be viewed as discrete dynamic systems. Typically, we describe the natural and social laws that govern material (i.e. physical, biological and social) systems in the form of CIM integrity rules (at the domain modeling level). Then, when we transform the domain model into an operational design, we formalize these rules in the chosen PIM language, after which they no longer refer to the material system itself but to its computational model. So, a PIM constraint refers to the state (and execution histories) of the software system that models (or represents) the material system under consideration

Rule R1 on page 7 is an example of a (deontic) static CIM constraint. An example of a (deontic) dynamic CIM constraint is: “it is obligatory that the confirmation of a rental reservation leads to an allocation of a car of the requested car group for the requested date prior to that date”. Well-known languages for expressing PIM constraint assertions are SQL and OCL. In logic programming, rules with empty heads (also called “denials”) corresponding to the negation of the conjunction of all body atoms are sometimes used as constraints.

The RuleML family of markup languages includes first-order logic languages which could be used to express constraint assertions. But RuleML does not provide any means to specify the constraint modality. It does also not have a clear separation of integrity rules from derivation rules. The latter deficiency is shared by SWRL.

Notice that a universally quantified implication with a definite consequent could be viewed both as a derivation rule and as an integrity rule. In order to avoid such ambiguities, we will strictly separate what is intended to be used as a derivation rule and what is intended to be used as an integrity rule in R2ML.

We first define the abstract concept of a *logical formula* in Figure 26. This concept becomes concrete, in the form of R2ML logical formulas, as soon as we specify the concrete R2ML syntax of atoms and of variable declarations. For atoms we can reuse the definition in Figure 22.

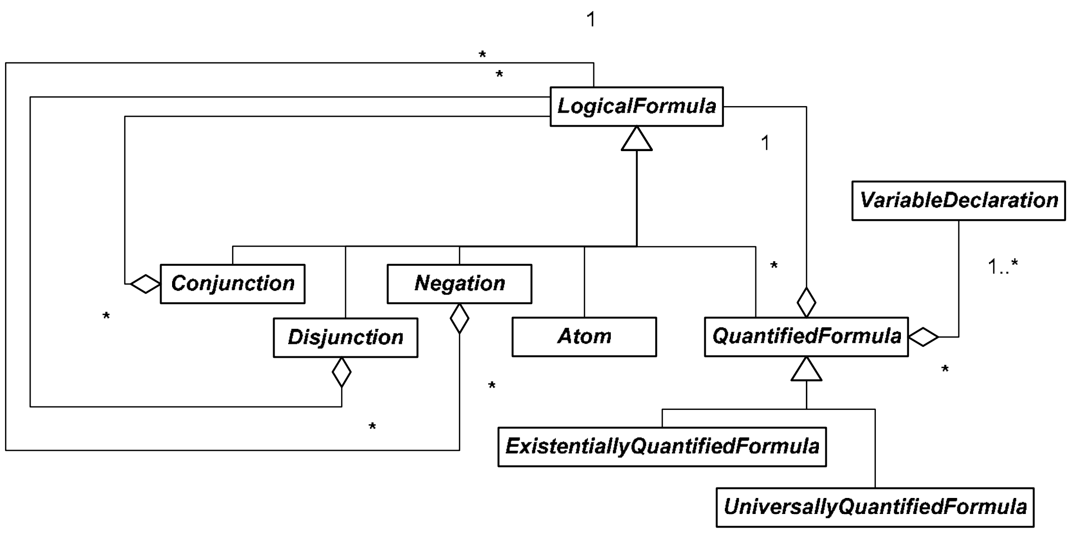

Figure 26: The abstract concept of a logical formula

A R2ML variable declaration is either an *object variable declaration* or a *data variable declaration,* as depicted in Figure 27.

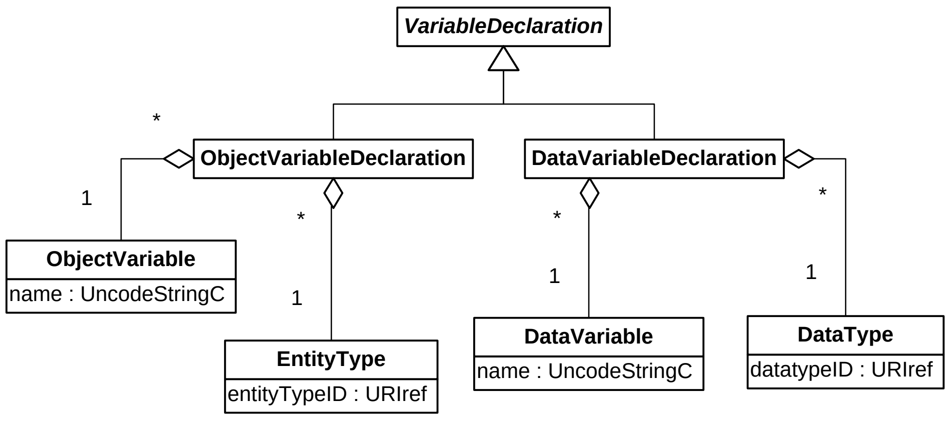

Figure 27: Variable declarations

Now we can specialize the above abstract definition of an integrity rule in Figure 25 by replacing the abstract concept of *LogicalStatement* with the R2ML concept of a logical statement as a R2ML logical formula with no free variables. In this way we have defined a general formula class that can be used as constraint assertions. It remains to be investigated in which way we can map OCL invariants to this R2ML constraint language. It is clear that data predicate atoms will be the most important R2ML formula for expressing OCL invariants.

5. Reaction Rules

Reaction rules are the second important type of rule in RuleML. Integrity and transformation rules have not received as much attention as derivation rules and reaction rules. Reaction rules are considered to be the most important type of business rule in [3].

Reaction rules consist of a mandatory triggering event expression, an optional condition, and a triggered action expression or a post-condition (or both), which are roles of type EventExpression, LogicalFormula, ActionExpression, and LogicalFormula, respectively, as shown in Figure 28. While the condition of a reaction rule is, exactly like the condition of a derivation rule, a quantifier-free formula, the post-condition is restricted to a conjunction of possibly negated atoms.

Action and event expressions may be composite and specified in different ways. For instance, the UML Action Semantics could be used to specify triggered actions in a platform-independent manner.

There is a little known parallel between derivation rules and reaction rules. Reaction rules are to dynamic (temporal logic) implication constraints what derivation rules are to static implication constraints.

There are basically two types of reaction rules: those that do not have a post-condition, which are the well-known *Event-Condition-Action (ECA)* rules, and those that do have a post-condition, which we call *ECAP rules*.

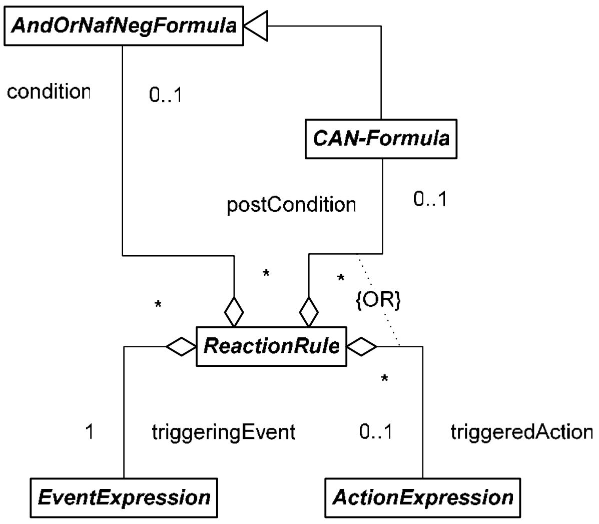

Figure 28: The abstract concept of reaction rules.

The post-condition of a reaction rule is either an atomic formula, a negation of an atomic formula or a conjunction of these. This is called a CAN-Formula in Figure 29. Such a definite formula specifies a deterministic update in a declarative way.

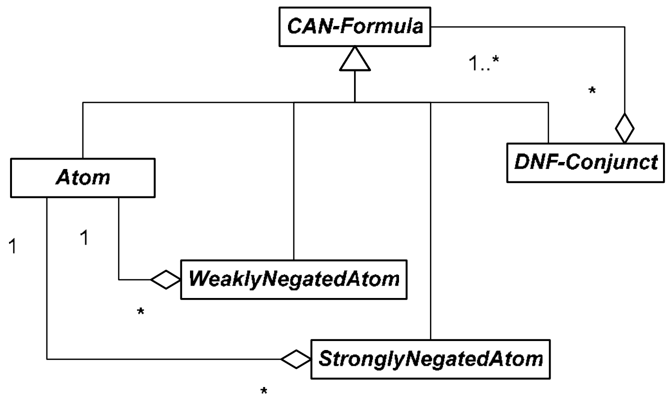

Figure 29: A CAN-Formula corresponds to a disjunctive normal form conjunct, that is, it is a conjunction of possibly (weakly or strongly) negated atoms.

By reusing the R2ML concept of atom, and-or-naf-neg-formulas as preconditions and CAN-formulas as post-conditions become concrete. However, we still have to define a concrete concept of event expression and action expression. For simplicity, we restrict our attention to the case of atomic events and actions, i.e. we do not consider composite event and action expressions.

Event types and action types are categories of entity types. Thus, they also have attributes and reference properties and we can reuse the concept of an object description atom from Figure 10, which we specialize to *event description atom* and *action description atom*. By replacing the abstract concepts of event expression and action expression with event description atom and action description atom, we obtain a concrete concept of reaction rule.

5.1. ECA Rules

An example of an ECA rule is R4 in Section 2. A restricted notion of ECA rules has been made popular in database research in the 1990s, especially by the work pioneered by Sharma Chakravarthy ([3]). These rules are now called database triggers in SQL. They refer exclusively to database state change events, and do not allow referring to general messaging events.

An application-specific ECA rule language may be used in software applications for handling application events in an automated fashion. A prominent example of this is the Microsoft Outlook rule wizard, which allows specifying email handling rules referring to incoming (and outgoing) message events.

5.2. ECAP Rules

Event-Condition-Action-Postcondition (ECAP) rules extend ECA rules by adding a postcondition that accompanies the triggered action. ECAP rules allow specifying the effect of a triggered action on the system state in a declarative manner, instead of specifying this state change procedurally by means of corresponding state change operations (like SQL UPDATEs). An example of an ECAP rule is the following:

> *Upon receiving an invoice requesting a payment of ?x USD, if the invoice is correct and the balance on the account is ?y and ?y is greater than ?x, then make a payment of ?x USD resulting in an balance of ?y - ?x.*

In this rule, the difference between the post-condition, expressed in OCL by

```
Balance = Balance@pre - ?x,
```

and the corresponding write operation, expressed in SQL by

```
update Account
set Balance := Balance - ?x
where AccountNo = ...
```

is rather subtle. But it should be clear that the former expression is a declarative sentence (which can be reasoned about), while the latter is not.

6. Production Rules

Production rule platforms are the rule technology that is most widely used in the business rules industry. Production rules consist of a condition and a produced action, which are roles of the type LogicalFormula and ActionTerm, respectively, as shown in Figure 30. While OCL could be used in a platform-independent production rule language to specify conditions on an object-oriented system state, the UML Action Semantics could be used to specify produced actions.

These rules have become popular as a widely used technique to implement 'expert systems' in the 1980s. However, in contrast to (e.g. Prolog) derivation rules, the production rule paradigm lacks a precise theoretical foundation and does not have a formal semantics. This problem is partly due to the fact that early systems used production/ECA-like rules, where the semantic categories of a rule's events and conditions in the left-hand-side, and of its actions and effects in the right-hand-side, were mixed up.

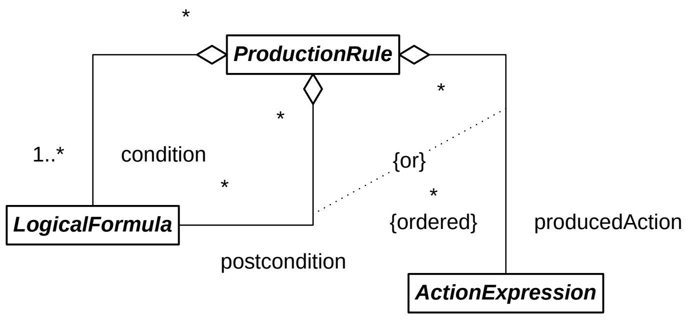

Figure 30: The abstract concept of production rules.

Production rules do not explicitly refer to events, but events can be simulated in a production rule system by externally asserting corresponding facts into the working memory. In this way, production rules can implement reaction rules.

A derivation rule can be implemented by a production rule of the form if-*Condition*-then-*assert-Conclusion* using the special action *assert* that changes the state of a production rule system by adding a new fact to the set of available facts.

The definition of R2ML production rules is obtained by replacing the abstract concept of logical formula with the R2ML concept of and-or-naf-neg-formula, and the abstract concept of action expression with the R2ML concept of action description atom.

7. Rule Languages for Other REWERSE Working Groups

Over the last three years, the RuleML initiative has produced several versions of its markup language family, with substantial changes and enhancements both of the syntax and the design rationale. The main concern has been to provide a more user-friendly syntax, maintain compatibility with RDF and support the integration of other logical languages, such as OCL and F-Logic. The modularization approach, needed for defining the RuleML language family in an integrated way, has been settled and upgraded to XML Schema. It is now flexible enough to accommodate various language extensions and variants. Several problems have been encountered during this process and were surmounted very recently. This effort gave some lessons on how to define and specify a general rule markup language. In particular, the syntax has changed several times while most of the concepts remained the same since the inception of the RuleML initiative. Our meta-modeling approach based on MOF/UML is concept-driven instead of syntax-driven. Consequently, it supports maintenance, modularity and reuse in the definition of syntaxes for specific user communities and for special purposes.

As it stands, RuleML defines the concept of logical derivation rules in the tradition of logic programming, including extensions for dealing with attributes ('slots') and two forms of negation. The RuleML initiative has now announced to provide extensions for supporting full First Order Logic and HiLog. Due to the more focused goals of REWERSE, work has developed at a good pace both at the syntactical and semantic levels, and the languages being proposed in the REWERSE working groups go beyond (and complement) the present capabilities of RuleML. This fact justifies the relevance and novelty of the work being made by REWERSE partners, in particular the potential impact and influence in the design of language(s) for the Semantic Web. To support this claim, we now analyze the languages being proposed and anticipated in the REWERSE working groups and relate them to the current RuleML proposal.

7.1. Working Group I2

Most of the policy languages surveyed by I2, and particularly, the proposed language PROTUNE, are based on normal or extended logic programming. The latest version of RuleML (version 0.88), is capable of representing logic programs with two forms of negation, with several add-ons, and therefore is able to represent most of object-level policies of PROTUNE. However there are several particular requirements of PROTUNE not being taking care of by RuleML, such as signed rules and handling of credentials.

A major problem with the current versions of RuleML is the incapability of associating meta-information with predicates and other constructs in the language. For instance, PROTUNE requires ways of declaring the sensitivity of predicates, literals and rules; categorize predicates, etc. Moreover, in order to represent metapolicies, PROTUNE also requires a language like HiLog to treat predicate names as syntactic entities; unfortunately, encoding of HiLog into RuleML is still underway. Furthermore, the meta-policy language requires built-in meta-predicates for term inspection, checking groundness, cryptographic functions, etc. In the current version of RuleML, only the list of built-in predicates defined by SWRL is supported. (see the discussion in the section on I3)

Since policy languages are used in P2P distributed systems, mechanisms are needed to specify and identify exactly the peers involved in querying and reasoning: the authority and context of PeerTrust language, or the actor of an action in PROTUNE. These mechanisms are not provided by RuleML.

In order to represent and reason with reputation-based trust, PROTUNE requires truth-values other than false and true (for instance integer confidence levels or values in the unit interval for representing probabilities), as well as 2nd order predicates for representing aggregate functions, like count, sum and max.

PROTUNE programs probably can be better viewed as special Event-Condition-Action programs, since it also resorts to the notions of actions which can be described and implemented, for instance, via imperative languages. For a more detailed discussion of the ECA extensions, the reader is referred to the subsection on I5 below.

There are still some other important issues mentioned by WG I2, but not yet included in the policy language, namely temporal constraints, priorities and conflict resolution, and complex actions. Libraries and language extensions are referred as important for simplifying the use and development of applications. These are mentioned here since they have been identified and planned to be addressed by other WGs, as we shall discuss below.

7.2. Working Group I3

RuleML does not provide any construct for software composition; not even the elementary import/export mechanisms of Prolog are supported. The supported typing mechanisms are also very basic. Both composition and typing are essential to a language for the Semantic Web. Regarding software composition, WG I3 illustrates the use of invasive software composition techniques which rely in the availability of a meta-language in the style of HiLog. It introduces the notions of fragment boxes, hooks and composers; all absent from RuleML. The software composition process is defined as transformations in meta-models. We suggest adopting the use of MOF/UML in order to simplify the integration with other Working Groups via the representations adopted by WG-I1.

RuleML can associate types with occurrences of variables, terms and individuals (logical constants). By associating individuals with types, RuleML has the capability in principle of representing descriptive typing. However, for prescriptive typing, RuleML lacks mechanisms for declaring signatures (even for basic types), in particular regular types used by the Xcerpt language being developed at WG-I4. The notion of pattern matching and constraint (in the sense of constraint logic programming) are also absent.

Built-ins for datatype predicates are defined outside RuleML, namely in the SWRL built-in namespace http://www.w3.org/2003/11/swrlb. Such a fixed list of built-in predicates is not sufficient for the purposes of REWERSE. Rather, a general rule markup language needs to support the unification of different languages for built-ins and need to be extensible (allowing the addition of new built-ins).

7.3. Working Group I4

The Xcerpt language rules can be interpreted both as a deductive rules as well as transformation rules. RuleML provides syntax for representing deductive rules and logical connectives (and, or and negation). These can be immediately used by a markup language for Xcerpt. However, Xcerpt does not distinguish between predicate and term symbols requiring new markup in RuleML, for syntactically representing data terms, query terms, and construct terms. Xcerpt also supports pattern matching with regular expressions, aggregations as well condition boxes defining semantic conditions using comparisons arithmetic expressions built over query variables. Once more, this is not supported by the existing RuleML proposal. Markup for allowing typing of Xcerpt programs is discussed in the subsection regarding WG I3; the XChange language is addressed in the next subsection.

Currently, Working Group I1 is constructing a MOF/UML modeling of the basic Xcerpt language in order to relate with RuleML and other languages being defined in REWERSE. The particular markup language should be extracted from the MOF/UML modeling, and must be immediately translatable to the language supported by the existing Xcerpt prototype. The current preliminary model is shown in the following figures. Some details are omitted, but most of the language has been modeled (the model is still subject to change). We follow a top-down approach in the presentation of the MOF/UML diagrams, and the reader is referred to the I4 deliverables in order to fully understand the language constructs and corresponding modeling.

Semistructured data processed and generated by Xcerpt are syntactically represented by Data Terms (see Figure 31). It is under investigation by I1 whether it is possible to represent Data Terms directly in RuleML without introducing new markup.

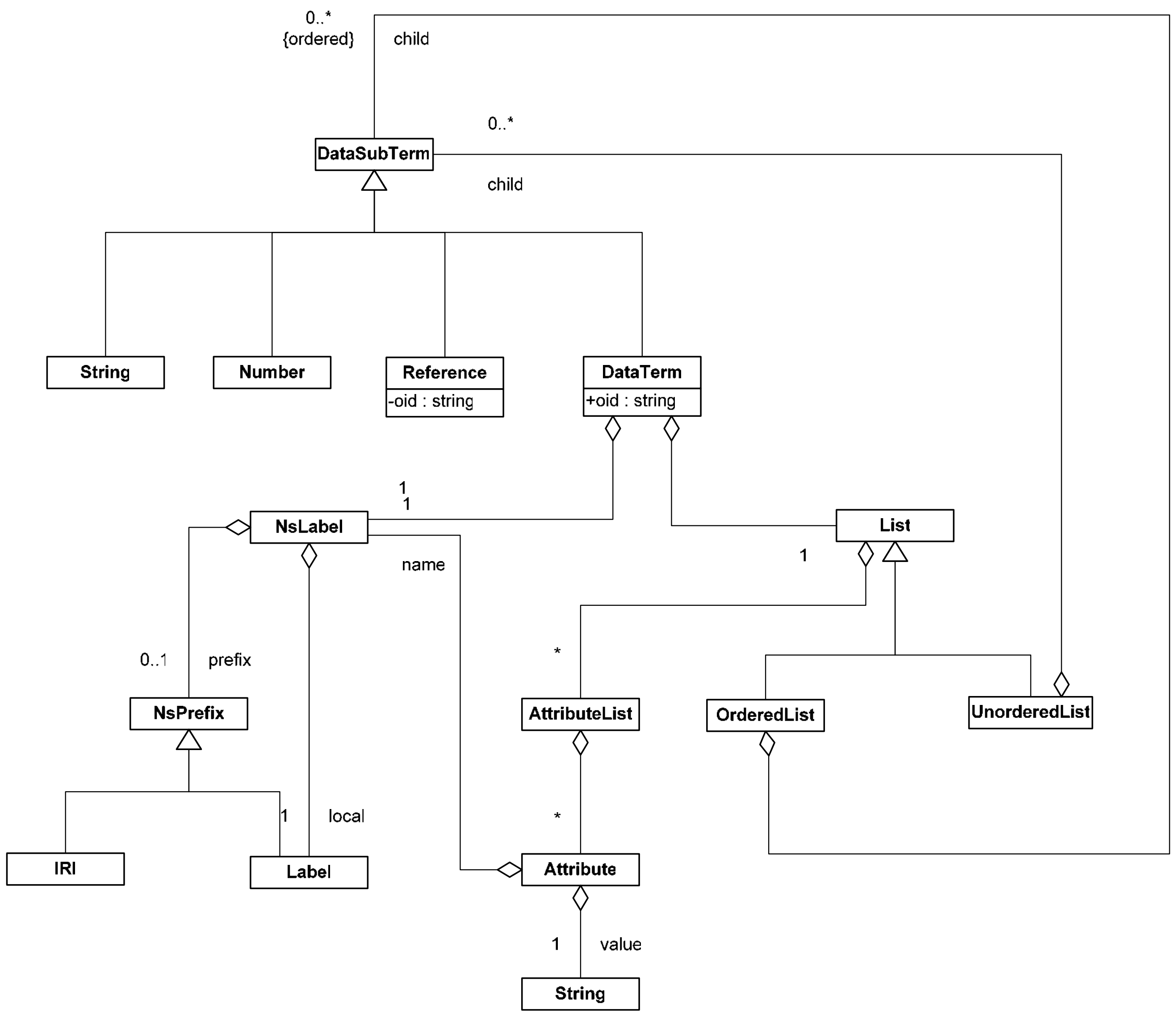

Figure 31: Xcerpt Data Term

An Xcerpt program consists of at least one construct-query rule and at least one goal. The head of construct-query rules and of goals are construct terms, while the bodies are queries (to be defined below). Notice that output resources can be associated with queries, and input resources can be associated with queries (input resources are encoded as Xcerpt data terms)

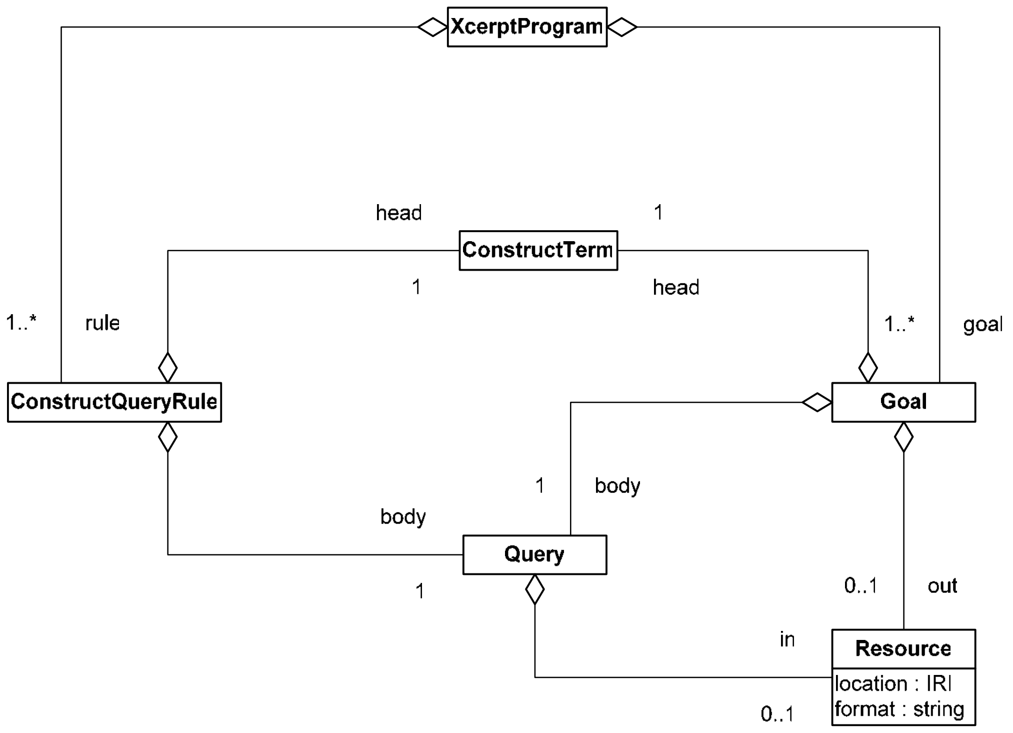

Figure 32: Xcerpt Program

The syntax of queries is described in Figure 33, and the usual Boolean connectives can be applied to QueryTerms. Queries may have an optional ConditionBox (comparisons on variables and arithmetic expressions). Modeling of ConditionBoxes is not yet provided.

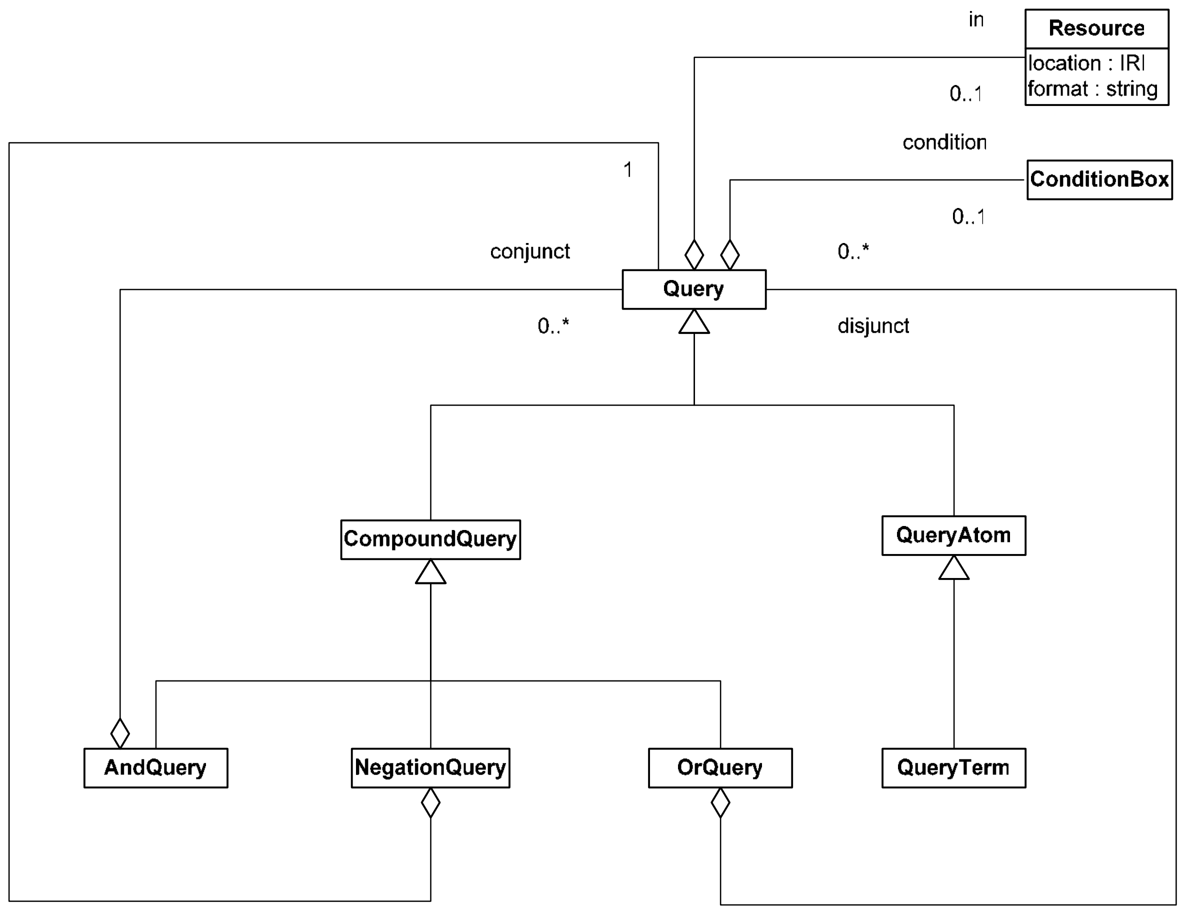

Figure 33. Xcerpt query

QueryTerms define matching patterns for input data terms, and can be quite involved. Variables, constants, and TermSpecifications are the building blocks of QueryTerms. TermSpecifications can describe both ordered/unordered as well as total/partial term matching patterns. Since they are more complex, they are presented in a separate figure below. Notice that variables can stand for term, namespaces as well as labels (tag names). There are other sophisticated constructs which are described in I4 deliverables (optional, without, descendant and position terms). RegularExpressions extend POSIX ones, and a diagram is not provided.

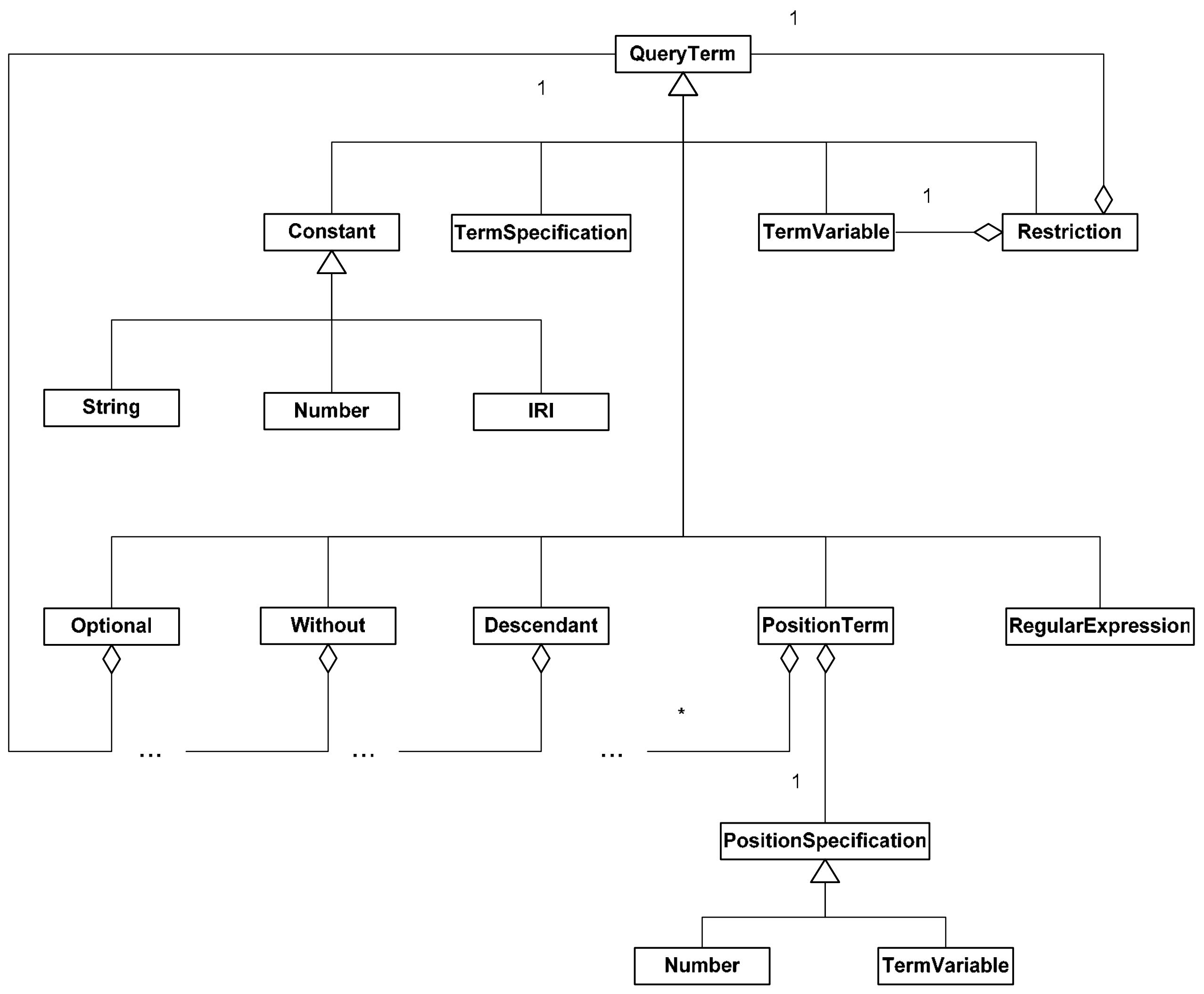

Figure 34: Xcerpt QueryTerm

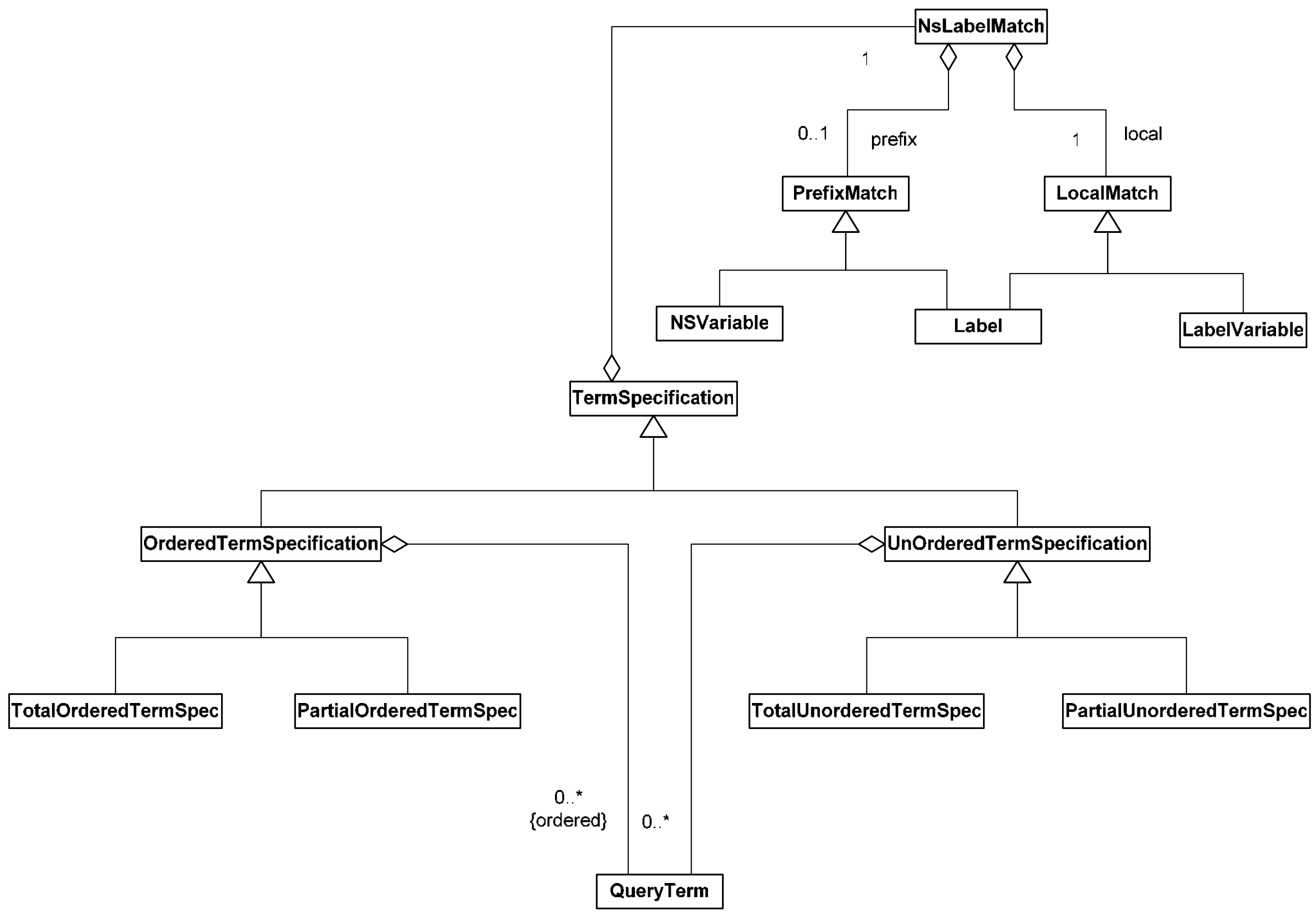

Figure 35: Xcerpt TermSpecification (a type of QueryTerm)

Construct terms are used in the heads of rules and goals to define templates to construct the generated output data term. Construct terms allow grouping and ordering mechanisms, as well as function expressions and aggregations. Partial term specifications are not allowed, since it does not make sense to generate partial data terms in output. Construct terms and TotalTerm-Construction are presented in Figure 36 and Figure 37.

These MOF/UML diagrams must be validated with respect to the existing Xcerpt prototype and aligned with RuleML language. It is evident that there is almost an isomorphism between the high level concepts of RuleML and Xcerpt. The details are however different, since for Xcerpt there is no distinction between Predicates and Terms. We plan to develop a mapping method to automatically generate the markup syntax from the MOF/UML model.

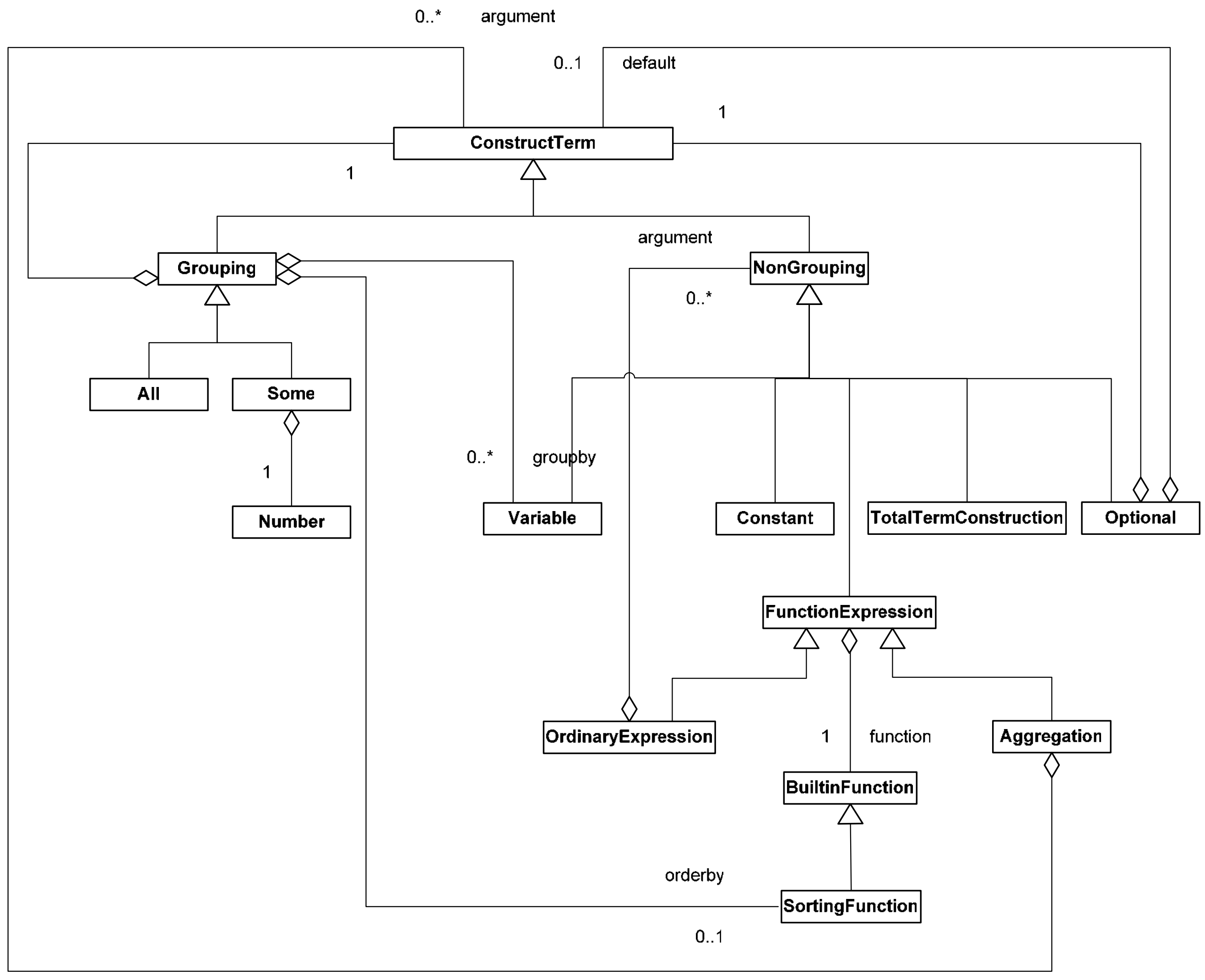

Figure 36: Xcerpt ConstructTerm

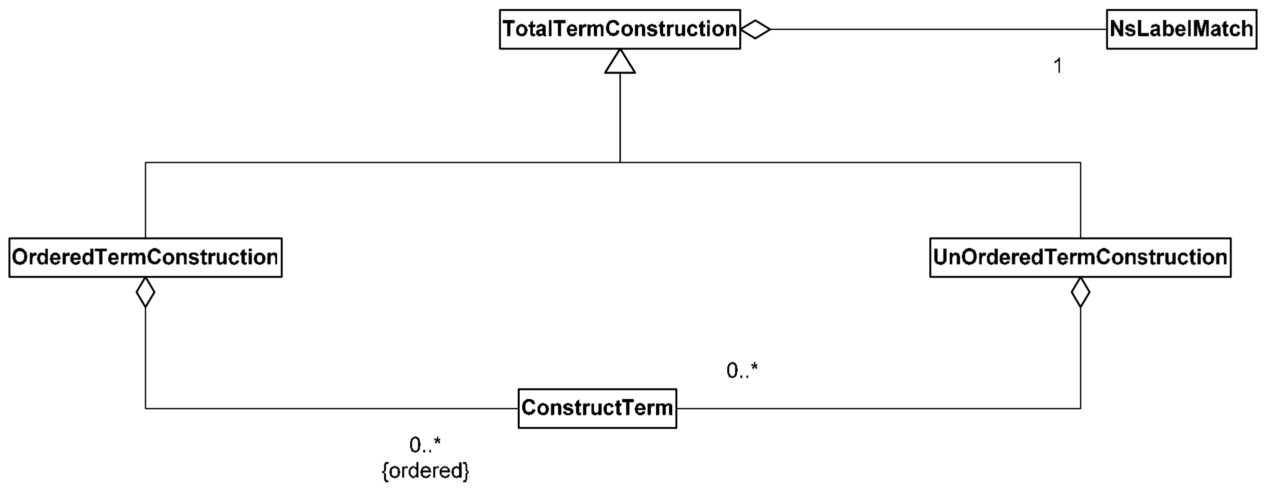

Figure 37: Xcerpt TotalTermConstruction

7.4. Working Group I5

The working group in evolution and reactivity is progressing on the notion of Event-Condition-Action(-Postcondition) rules. Syntax for this form of rules is not yet available in RuleML, being identified the high-level components of this form of rules, as described previously in Section 6. The requirements for each component of the ECA(P) rules are now listed.

Working Group I5 described several notions of events, and resorts to event algebra to combine them; this helps in the RuleML definition of EventTerm. In particular, atomic events in the XML tree should be provided based on the DOM Level2/3 events. This requires a way of internalizing a XML document in logical form. We suggest to consider the use/adaptation of the RDF Schema for XML Infosets (see http://www.w3.org/TR/xml-infoset-rdfs and http://www.w3.org/TR/xml-infoset/) for a logical representation of an XML document.

The definition of conditions are not a direct concern of WG-I5, but it is assumed that conditions might query one or more peers, and communication is peer to peer wrapped in XML, via messages and events. This requires the same mechanisms described by I2 for addressing, identifying and querying remote peers, in particular global ECA rules. It also requires a way of accessing the information before and after an action took place. This has not been addressed by the RuleML initiative.

Finally, in the action part of ECA rules, the capability of extending the system with new elementary actions, as well as mechanisms for combining these actions into complex transactions is required. Again, these issues have not been tackled by the RuleML initiative, and are used for instance in the XChange language

Since, as planned, WG-I5 has only presented use-cases it is not possible at present to define a concrete language model for WG-I5 ECAP rules.

7.5. Working Group A1

The Geo-Temporal specification language GeTS defines a functional language which is able to describe and manipulate temporal notions. This fundamental work defines several datatypes and functions which can be used in the rule based languages being defined by the previous Working Groups for performing temporal reasoning; this is particularly important for Working Groups I2 and I5. From the point of view of a rule markup language the GeTS specification language is simply defining a module of built-ins that can be used in the rule language. The basic mechanisms required from the rule markup language are the notions of types, modules, and support for built-ins, none of them available in RuleML. This is expected to be developed in I2, and therefore the markup language should be immediately applicable to the particular case of GeTS, and this provides a significant use-case for I1 and I2. It is worth mentioning that it could also be possible to use (fuzzy) time intervals as non-boolean "truth-values" in a rule language, as the ones being studied in I1.

7.6. Summary of Features Lacking in RuleML

In the following table we list the most important features necessary for the definition of the REWERSE languages lacking in the current version of RuleML. We distinguish for each Working group the features which appear to be fundamental (marked with 'X') and the ones which might be important but not prioritary. When known, we also indicate the Working Group being responsible for defining the language supporting the feature. This table is subject to future revision and upgrading.

	I2	I3	I4	I5	A1	Defined by
Meta Declarations	X	X			*	
Higher Order Language	X	X				
Built-ins	X	X	X	X	X	
Non-boolean truth-values	X				*	I1
Aggregation	X		X			
Peer addressing	X			X		I2
Temporal Reasoning	*				X	A1
Defeasible Reasoning	*	*				
Modules	*	X			X	I3
Transactions/Complex Actions	*		X	X		I5
ECA(P)	*			X		I2
Types		X	X		X	I3
Events/Event algebra				X		I5
Pattern matching		X	X			I4
Transformation Rules		*	X			I4

8. Conclusion and Future work

In this report, we have presented a rule markup language design approach based on the method of MOF/UML meta-modeling for defining the abstract syntax of a language. We have analyzed all important concepts of OWL, SWRL and RuleML. On the basis of this analysis, maintaining compatibility with OWL, SWRL and RuleML, we have defined metamodels for vocabulary elements, derivation rules, integrity rules, reaction rules and production rules. These metamodels define our language R2ML. It is clear that we can obtain concrete markup languages by mapping them to corresponding XML schemas with help of a general mapping procedure to be developed as the next step in our research line. In this way, WG-I1 will obtain a comprehensive rule language framework that provides a markup syntax for all kinds of computational logic rules and that will be extended and adapted according to the needs of the other REWERSE working groups.

References

[1]Boley, H.: The Rule Markup Language: RDF-XML Data Model, XML Schema Hierarchy, and XSL Transformations, Invited Talk, INAP2001, Tokyo, Springer-Verlag, LNCS 2543, 5-22, 2003.

[2]Boley, H., Tabet, S., Wagner G.: Design Rationale of RuleML: A Markup Language for Semantic Web Rules. International Semantic Web Working Symposium (SWWS), June 2001, Stanford, USA.

[3]Taveter K., Wagner, G.: Agent-Oriented Enterprise Modeling Based on Business Rules. In Proc. of 20th Int. Conf. on Conceptual Modeling (ER2001), Springer-Verlag, LNCS 2224, pp. 527–540, 2001.